

Digital Connection

Don Rotolo, N2IRZ

Sketching an Arduino

This document ©2025 by Don Rotolo N2IRZ under CC BY-NC 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc/4.0/>.

This document was donated to the Digital Library of Amateur Radio & Communications (DLARC) for public access. It differs slightly from what appeared in *CQ Amateur Radio Magazine*.

Originally in CQ Magazine, February 2011

In December's column, I mentioned my MPGuino, a fuel economy gauge built around an Arduino platform. This month, we'll take a closer look at the Arduino development boards and map out a project.

In past columns, we've established that fewer hams are building things these days. My assertion is that while this may be true for hardware, more people are getting their creative juices flowing with software than ever before. There are dozens of nice microprocessor platforms available for experimenters, each having their advantages, but for this project I've selected the Arduino Uno board.

Arduino is an open-source hardware and software platform developed in Italy, assembled around **A t m e g a 1 6 8** processors. Other systems I considered were the BASIC Stamp and the Propeller (from Parallax) and the Rabbit (from Rabbit Semiconductor). All of these platforms (and a few others I haven't mentioned) are well-supported and have an easy-enough programming process that caters to beginners, I don't think you can go wrong with any of them. But, being of Italian descent and wanting to

support open-source projects, Arduino seemed like a nice fit.

The Design Process

The first step in any design is to clearly identify the problem you are trying to solve. This is more important than it seems, since a solution to a different problem will be disappointing. In my case, I had built a massive antenna rotator (being far too cheap to actually buy a commercial unit) and needed some way of controlling it.

What I built is based upon a 33:1 worm-gear right-angle reduction drive with one horsepower capacity, bought on eBay for \$75 including shipping. I then found a power window motor from a high-end luxury car to drive it. The motor came from a car that had auto-up and auto-down windows, using a pair of Hall-effect sensors in quadrature to measure speed and direction, all operating on 12 volts of course.

A Hall-effect sensor is a solid-state device that senses magnetic fields. In my motor, two of them are mounted next to a magnetized toothed wheel in such a way that one of the sensors detects a 'tooth' before the other. One sensor can be used to count teeth, thus deriving motor speed and position, while the second sensor can be used



An Arduino Uno board and motor driver shield. At this point, I need to solder the stackable header connectors to the shield so it can be set atop the Uno. During development, the Uno board is powered by the USB cable, which is convenient. Note the yellow LED lit up near Pin 13 (to the upper left of the Arduino symbol), a result of running the "Blink" example sketch (see text).

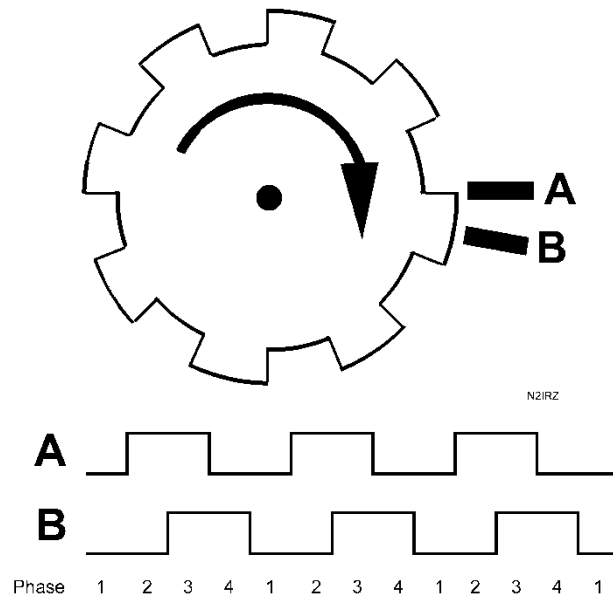


Figure 2: A rotary encoder uses two sensors arranged in quadrature to sense rotation direction. Since sensor A sees the gear tooth before sensor B, it must be rotating clockwise. This gear is shown just at the end of Phase 3.

with the first to determine the direction of rotation. Looking at Figure 2, if sensor A sees the tooth before sensor B, rotation is clockwise, while if B comes before A, rotation is counter-clockwise. The sensors in my motor have drive and output circuitry that delivers a nice, clean square wave, simplifying the interface.

I fabricated a shaft adapter so the motor could securely turn the worm drive input shaft along with a simple motor mount. I bought a used antenna rotator at a Hamfest for \$10 (not working) for the mast adapter casting, and the result is shown in Figure 3.

Now that we know what problem I need to solve – the basic purpose of my project – the next step in the design process is to make a list of the functions or capabilities my device requires. First, I need to be able to power a 12 V DC motor that draws up to 2 Amps under load (I measured it) both forward and backwards. Second, I need to be able to read two Hall-effect sensors, specifically counting the number of pulses from one of them, and detecting the leading edge of the output pulses from both of them. It is not allowed for any pulse or leading edge to be ‘missed’: The counting must be 100% reliable. Third, I need a way to control the motor from my shack, and last I need some kind of display to show where the antenna is pointing.

Implementing a Design



Now we need to figure out exactly what we’ll need to implement those capabilities. We know we have a microprocessor, so the inherent capabilities are there. To drive the motor, I need some kind of motor driver, since no MPU can drive a motor directly. Sparkfun sells a Motor Shield for the Arduino, with 2 channels of variable output in both directions, capable of handling 2 Amps per channel. I suppose I could have built a motor driver (using, for example, a National Semiconductor LMD18201 chip) but for \$25 I bought the off-the-shelf solution and used only one of the two drivers.

I need a 12 V DC power source for the motor, so I’ll tap off shack power that already exists at the base of the tower, and for the 5 V DC to power the Arduino board, I’ll use a simple LM7805-based power converter.

To read in the motor speed and direction, I’ll just use the nice, clean square wave outputs from the Hall-effect sensors, brought down the tower via shielded cable. The Sensors also need 12 volts and ground to operate, which will be sent up the tower in the same wire bundle. However, a significant issue is the 100% reliability requirement: Just using the analog inputs to the Arduino won’t guarantee that

Figure 3: My super-duty homebrew antenna rotator uses a car power window motor, an old rotator’s mast mount casting and a super heavy-duty worm gear reduction drive. I’m using an Arduino Uno board to control the motor and sense where the antenna is pointed.

pulses will not be missed. The answer is to use Interrupts, which I'll explain in a moment.

A display of the antenna direction can take several forms. The easiest to implement would be a circular pattern of eight LEDs, one or two lit to show (within 22 degrees) where the antenna was pointing. But, since I'll have access to the motor pulse count, I'll know the exact direction down to a fraction of a degree, so I chose to use an LCD display to show the direction in degrees. (In reality, the antenna system isn't mechanically sensitive enough to adjust in fraction-of-a-degree increments, but we'll gloss over that inconvenient fact).

Controlling the whole contraption will take the form of two buttons: clockwise and counter-clockwise. Simple and reliable. Maybe in the future I'll develop a set & forget system where I can dial in the direction and let the system take care of movement, or perhaps even interface with a logging program for completely automated pointing, but for now, simple is where we're headed.

The Details

The last step of the design is to write out the specific hardware and software details. Unfortunately, if we were to get deeply into that, I'd need several times more space on these pages than my editor can offer. So instead, let's cover some of the background for those of us who are fairly new to all this.

Arduino Platforms

The Arduino comes in several types (see <http://arduino.cc/en/Main/Hardware>), depending on what exactly you need. The Uno board is the latest revision of their general-purpose development board, replacing the Duemilanove (meaning "2009" in Italian) board, which is still available and also not a bad choice. If you need more processing power, memory and input/output pins, they have a few versions of their Mega board. The Diecimila ("10,000") runs on 3.3 V instead of 5 V, and the innovative Lily Pad is a stripped-down version with a circular circuit board, intended to be sewn into clothing for wearable processing applications. The Nano is very small, the Fio and Bluetooth are intended for wireless applications, and the Pro and Pro Mini are bare-bones full-power systems intended for embedded

applications. There are several other variations on these, as well as many older and out-of-production boards, so have a look at the web site for complete details.

Peripherals

Also available, from both Arduino and several others, and so-called "Shields". These are daughterboards that provide specific functions. For example, I bought the Motor Shield so I could easily control my rotator motor. Other examples include an Xbee Shield for wireless communication using Xbee, an Ethernet Shield for connecting to the Internet, and a Sensor Shield to accommodate several different available sensor sub-boards to sense just about any physical thing there is (temperature, pressure, light, distance, etc.). Search on eBay for "Arduino Shield" to see what I mean – just remember Caveat Emptor when buying items from overseas.

Programming

One nice thing about the Arduino is the "Learning" section of their web page. After getting my Arduino Uno in the mail, I downloaded the programming environment from the Arduino download page (<http://>

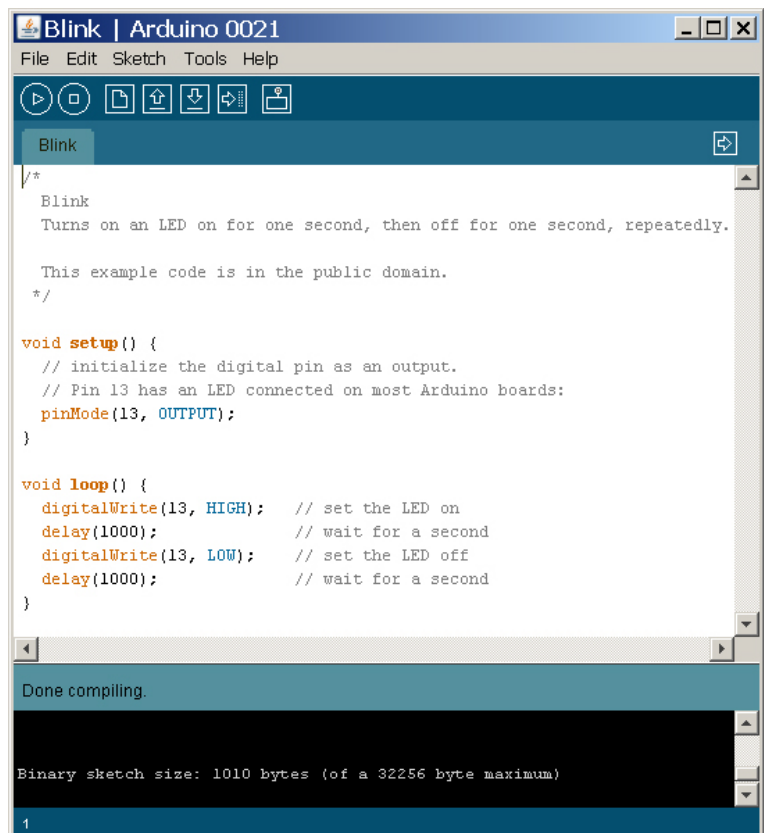


Figure 4: The sample "Sketch" (program) called "Blink" makes the LED connected to Pin 13 of my Uno board blink on and off every second (1000 milliseconds). Several example Sketches, along with many online resources, ease the task of learning how to program and Arduino.

arduino.cc/en/Main/Software>) and unzipped the package into its own directions, making sure I preserved the directory structure. I started the programming environment, selected the type of Arduino board I had, connected it up (via USB, which also powers the board), and started following the tutorials.

My first "Sketch" (a program for the Arduino is called a Sketch) had the on-board LED blinking once per second (Figure 4). I fooled with that for a while, changing the blink speed and reloading it onto the board each time to see what would happen. I then realized that during the 1000 millisecond (1 second) delay period, the processor couldn't do anything else - it was focused on waiting - so I then tried the Blink Without Delay sample Sketch, which approached the problem differently: Instead of waiting, it would periodically check the value of the on-board timer and when it increased by 1000 milliseconds, caused the LED to blink. By doing it this way, other tasks could run while we were waiting for the next second to start.

This simple example starts to hint at the nuances of programming. While both examples could blink an LED every second, the later example did it while preserving the ability to perform other tasks.

Now on to my task-specific Sketch. Controlling the motor was actually very simple: I'd read in the values of both switches (which switched a 10k Ohm pull-up resistor to ground) and turned the motor depending on which switch was pressed. If both switches were pressed, I'd stop the motor, just as if neither switch was pressed. Simple enough, right? But then the problems started.

While a switch was being pressed, the processor focused on that task alone, and couldn't detect the pulses from the Hall-effect sensors. So, I decided to use Interrupts. An interrupt is a hardware-based signal that actually interrupts the program, no matter what it is doing at the moment, to perform some task while the interrupt is active. Of course, I couldn't interrupt the program for long, or things would stop happening, so I wrote some code that, when the interrupt became active, added (or subtracted, depending on the motor direction) "one" to a counter and popped immediately back into the main program. This took only a few milliseconds (if that) so the main program ran smoothly despite the interrupt. I used two counters, one for each sensor, although one would have been enough.

I also added some logic to the counting module to determine which interrupt line (there was one for each Hall-effect sensor)

became active first, and stored that information as "direction". (I also verified that measured direction was the same as that commanded by the buttons. If there was a discrepancy, the motor was stopped).

For the display, I included a module in the programming library that made using an LCD display very easy: You just "write" the text value to a certain output line and it magically works. In each run-through of the main Sketch (the program runs continuously in a loop) I read in the value of counter #1, divide by a number corresponding to the number of pulses to turn the rotator output shaft by one degree, round off the value to a whole number, and Write that to the display.

The last task was some error management: I didn't want the rotator to turn more than about 1.1 rotations, so I put in some code to stop the motor and disable the switch for that direction when the count increased to a number representing 1.1 turns, or decreased to zero. I also decided to put in a limit switch at the zero point - whenever the rotator reached zero, the counter would reset to zero no matter what it actually was. I also put a note on the display whenever this happened, since it meant that my counting was not working properly.

Some might argue that I really need a brake function to protect the worm gear, but I haven't figured out how to do that on the rotator. That's another task for the future, just like actually mounting it to the tower, weatherproofing it and installing an antenna. I'm waiting for the next ice storm for those tasks...

Conclusion

That's Arduino in a nutshell. To get started, first go buy an Arduino (visit <http://arduino.cc/> for a list of distributors) , download the programming environment, and start playing with it. Make LEDs blink, read in switches and potentiometers (there's an example Sketch for that), and start dreaming of some practical application. Start off simple & easy (e.g., don't try to build a TNC), make use of the help resources such as the online forums (where others are happy to help) to make things happen. Do that, and soon you'll be breadboarding in a new way.

Until next time, when I'll share what David Levine K2DSL taught me about working the International Space Station, happy programming!

73 de Don N2IRZ.